A Tale of Two Dinosaurs: Creation and Evolution of a Simple Graphics Renderer

James Russell
Software Engineer

Sony Computer Entertainment Europe

Teal

SONY

COMPUTER
ENTERTAINMENT

The Next Generation PlayStation Developers Seminar

James Russell

Technology Group
Sony Computer Entertainment Europe

A Tale of Two Dinosaurs - June 1999

Confidential

A Tale of Two Dinosaurs

- A simple introduction to the concepts involved in getting a high performance renderer operating on the next generation PlayStation

Building the renderer

- Similar to the PlayStation code
  - Use the original T-Rex model
  - Uses 3 parallel lights + ambient
- Model is not textured
  - Renderer speed is governed by calculation time, not drawing time
  - Sample code is easier to write
  - Textured render code has negligible speed difference (1%-2%)

A Tale of Two Dinosaurs - June 1999 Confidential

In this presentation we will describe the process of building a simple real world renderer for the next generation PlayStation. The purpose of this is to show a practical example of the capabilities of the machine rather than just quoting the maximum figures, and also to show some of the stages involved in optimising performance for this machine.

To aid comparisions, the renderer that I will build will operate in the same way as a renderer on the original PlayStation. So the model will be transformed with 3 parallel lights and an ambient light. The model will be a non textured version of the original T-Rex that was featured in the original technology demos for the original PlayStation. No textures will be involved to simplify the operation of the renderer for this presentation.

The T-Rex model

- Original PlayStation model contains 2724 triangles.
- Original runs at 60Hz, giving around 500,000 vert/s on the PlayStation.

Rendering components

- EE core used to move data and build display lists
- VU0 used as 'GTE+'
- VU1 used as an independent transform engine
- GS Graphics Synthesiser

The core of the next generation PlayStation consists of two chips.

The first chip, the Emotion Engine, consists of a MIPs compatible CPU and 2 vector processing units VU0 and VU1. The CPU is a 2 way super scalar design, and can use VU0 as a coprocessor (in the same way as the original Playstation used its Geometry Transformation Engine (GTE), but much more flexible).

As well as I-cache and D-cache the CPU also has a scratchpad memory system, which operates at full speed. This memory can be used by DMA for transfers to other devices or to/from main memory - it is one of the key elements that needs to be utilised to obtain the most from the system.

The vector units operate on 4x32bit floating point vectors, and have built in instruction memory and data memory. Vector unit 0 is connected to the CPU, and can be used in macro mode - where individual instructions are executed as CPU coprocessor operations, or micro mode - where small programs are executed independently without involving the CPU. Vector unit 1 is operated entirely in micro mode, and has no direct connection to the CPU. Each vector unit is connected to main memory by a DMA channel, and vector unit 1 also has a direct connection to the GS interface.

The other chip is the graphics synthesiser. This operates on drawing commands sent from the EE and will draw triangles, triangle strips and fans, lines and sprites.

"My first renderer"

- Model is processed in groups of 40 triangles ⇒ 120 vertices per group
- Data stored in similar format as original PlayStation model
  - vx,vy,vz 16 bit 1.11.4 fixed point format
  - nx,ny,nz 16 bit 1.3.12 fixed point format

A Tale of Two Dinosaurs - June 1999 Confidential

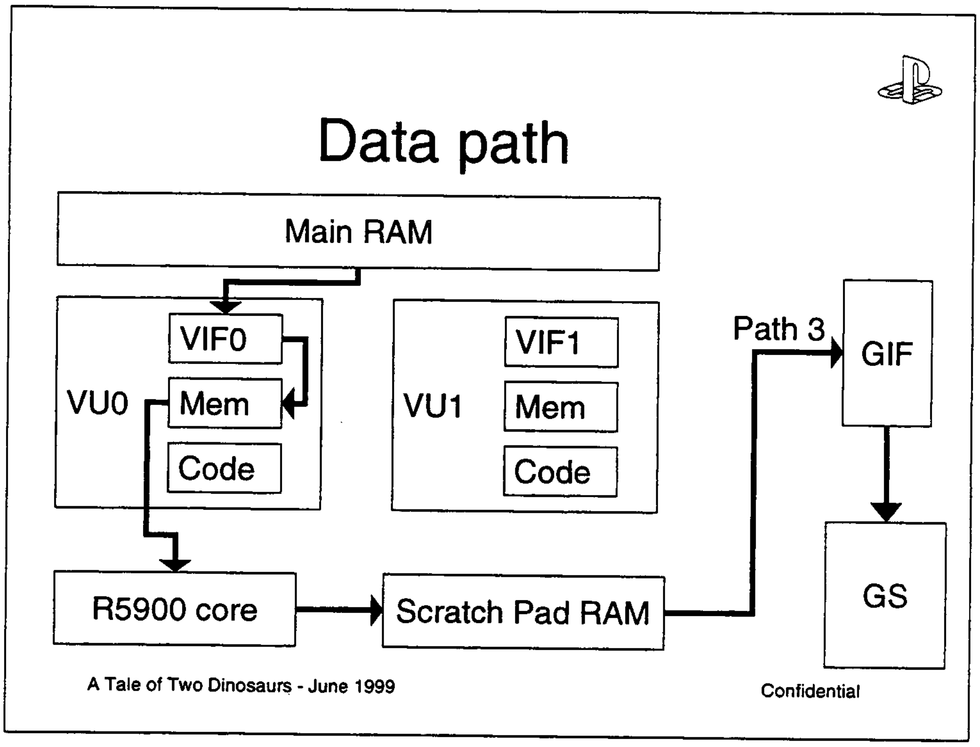
On the original PlayStation the T-Rex model was stored in Sony's TMD format. With this format the normals and vertices are stored in separate tables and indexed in the polygon descriptions.

For the next generation PlayStation the use of DMA is far more important for transferring data into the vector units as well as transferring polygon primitives to the GS, so a different format had to be used. For lighting a normal per vertex is needed so the data format groups point info into pairs of vertex and normal.

These vertices are stored in main memory as 16 bit values, but need to be 32 bit values for the vector units to operate on. (We could store the model using floating point vectors, but in most games memory constraints apply.) The DMA channel to load data into the vector unit supports bit expansion however, so the expansion can occur transparently. The conversion from 32 bit integer vectors to floating point vectors still needs a vector unit operation however.

Operations involved

- Colours
  - Rotate normal to world
  - Clamp to [0.0 - 1.0]
  - Apply to colour matrix to give vertex colours
- Vertices
  - Apply vertex to LocalToScreen matrix


Data path
Main RAM
VU0
VIF0
Mem
Code
VU1
VIF1
Mem
Code
Path 3
GIF
GS
R5900 core
Scratch Pad RAM
A Tale of Two Dinosaurs - June 1999
Confidential

Program flow

- Use VIF0 to unpack data
  - VU operates on 4x32 vectors only
- Use SCE supplied library
  - Provides basic geometry transform functions
- GIF packs results for GS
  - Build block on scratchpad RAM,then transfer to GIF using DMA

A Tale of Two Dinosaurs - June 1999 Confidential

The core program for the 1st render is shown below. It consists of the DMA read from main memory to Vu0 memory, transformation using the sceVu0 functions, and then the DMA write from memory to the GIF.

```
sceDmaSend( dmavif0, spad_vif0 );   // Send block of 120 vertices to VU0 ram
sceDmaSync( dmavif0,0,0 );          // Wait for DMA to end
pi = VU0_MEM;   gi = buffer;        // Process from VU0 ram to primitive buffer

for ( i=0;i<(40*3);i++) {
  sceVu0ITOF12Vector( gi,pi );              // Convert normal to floating point vector
  sceVu0ApplyMatrix( gi,local_light,gi );   // Calculate light influence
  sceVu0ClampVector( gi,gi,0.0f,1.0f );     // Limit lights
  sceVu0ApplyMatrix( gi,light_colour,gi );  // Apply colours
  sceVu0FTOI0Vector(gi, gi);                // Convert final colour to integer vector
  sceVu0ITOF4Vector( &gi[4],&pi[4] );       // Convert vertex to floating point vector
  sceVu0RotTransPers( &gi[4],local_screen,&gi[4],0 ); // Transform to screen XYZ
  pi += 8;   gi += 8;  }                    //  Each vector is 4 words long

sceDmaSend( dmagif, spad_gif);      // Send block of 120 vertex/colour pairs to GS
sceGsSyncPath( 0,0 );               // Wait for DMA to end
```

V1.0 Performance

- T-Rex is 2720 tris, or 8160 vertices
- Version 1 draws T-Rex in 75 scanlines
- Performance of the first renderer is around 1.73 Mvert/s
  - Not very good!

A Tale of Two Dinosaurs - June 1999

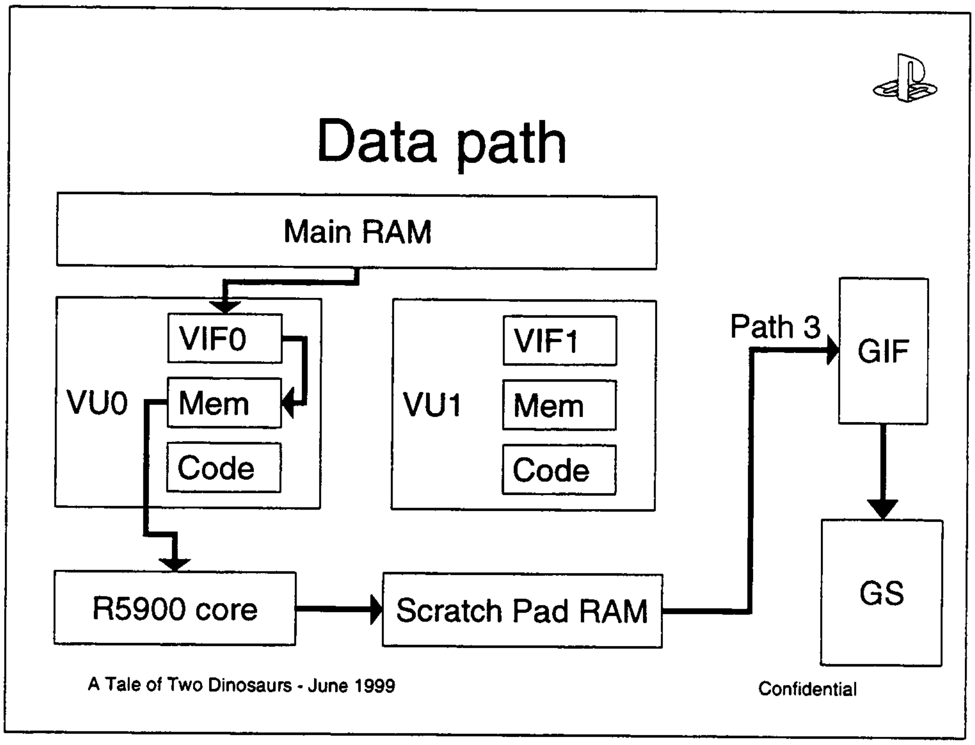
Confidential

My 2nd renderer

- Inline the SCE VU0 library functions into the render loop
  - We don't have to reload the matrices
  - Save on some load / store pairs

For the second version of the renderer code the Library functions are replaced by their inline assembly code directly. (This was done by simple cut&paste from the library source.) The code was then rearranged slightly as shown below:

```
asm __volatile__("
la $8,0x11004000  #VU0_MEM
la $9,buffer                     #GIF buffer
ori $10,$0,40*3   #40 Triangles
_calc_vertex:
lqc2 vf4, 0x0($8)                # Load Normal
#  sceVu0ITOF12Vector(gi, pi);   // Normal
vitof12.xyzw vf8, vf4
#  sceVu0ApplyMatrix(gi, local_light, gi);      // Light influence
vmulax.xyzw      ACC,  vf16,vf8
vmadday.xyzw ACC,  vf17,vf8
vmaddz.xyzw      vf3,  vf18,vf8
lqc2 vf2, 0x10($8)               # Load Vertex
#  sceVu0ClampVector(gi, gi, 0.0f, 1.0f);       // Limit lights
vmaxx.xyzw $vf03,$vf03,$vf0      # v0.x is 0.0
#  sceVu0ITOF4Vector(&gi[4], &pi[4]);           // Point
vitof4.xyzw      vf8,vf2
vminiw.xyzw $vf03,$vf03,$vf0     # v0.w is 1.0
```


Data path
Main RAM
VU0
VIF0
Mem
Code
VU1
VIF1
Mem
Code
Path 3
GIF
GS
R5900 core
Scratch Pad RAM
A Tale of Two Dinosaurs - June 1999
Confidential

V2.0 Performance

- Version 2 draws T-Rex in 43 scanlines
- This is almost 2x version 1 speed
- Performance is now nearly 2.8 Mvert/s
  - Better, but nowhere near theoretical limits

A Tale of Two Dinosaurs - June 1999

Confidential

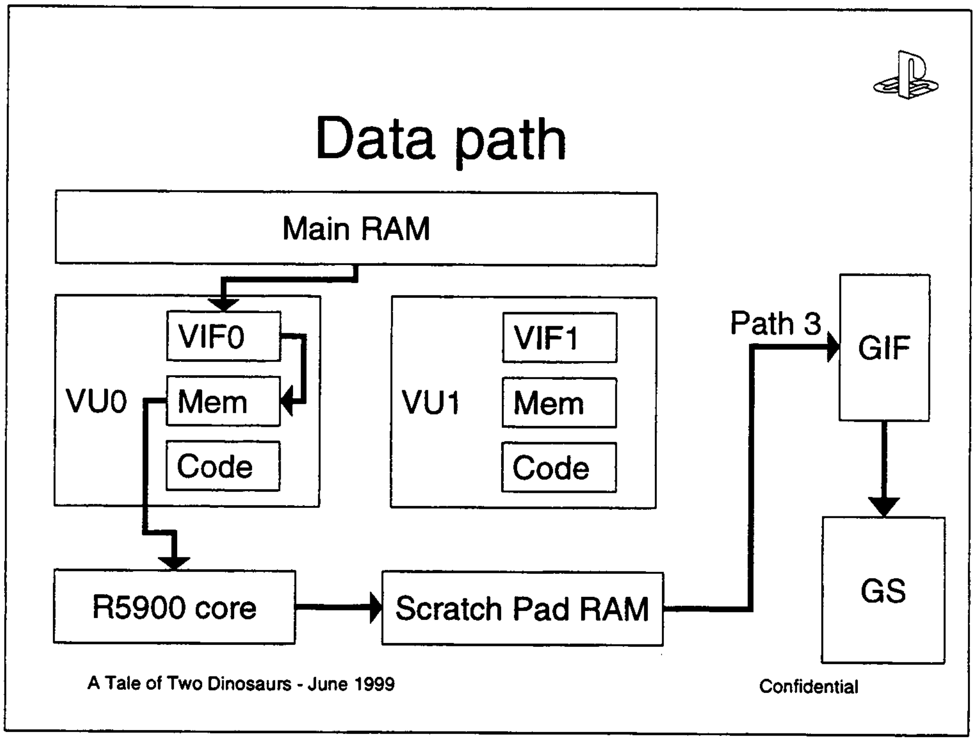
My 3rd renderer

- Render routine is rewritten as VU0 micro subroutine
- This allows overlap of EE core code and VU0 code
- VU0 micro program fetches vertices & normals directly from VU0 RAM

A Tale of Two Dinosaurs - June 1999   Confidential

For this version of the renderer the main CPU code for the core loop is a lot simpler as it only has to store the results for each vertex. After each vcallms operates the result is read from the VU0 register (Note that the qmfc2 instruction has a .i option to wait for the microcode to complete).

```
asm __volatile__ ( "
la $9,buffer          #GIF buffer
vcallms vu0_first     #Calculate first point
qmfc2.i $10,vf4                   #Read colour
qmfc2 $11,vf5                     #Read vector
or $8,$0,119
_calc_next:
vcallms vu0_next      #Calculate next point
sq $10,0($9)                      #Store previous colour
sq $11,0x10($9)                   #Store previous vertex
qmfc2.i $10,vf4                   #Read colour
qmfc2 $11,vf5                     #Read vector
subu $8,$8,1
add $9,$9,0x20
bnez $8,_calc_next
sq $10,0($9)                      #Store final colour
sq $11,0x10($9)                   #Store final vertex
": : : "$8","$9","$10","$11" );
```


Data path
Main RAM
VU0
VIF0
Mem
Code
VU1
VIF1
Mem
Code
Path 3
GIF
GS
R5900 core
Scratch Pad RAM
A Tale of Two Dinosaurs - June 1999
Confidential

```
nop                          div Q,vf0w,vf5w    ;Calc 1.0/w for perspective
maddaz.xyzw ACC,vf22,vf3     nop
maddw.xyzw vf4,vf23,vf0      nop                ;( W=1.0 for ambient )
ftoi0.xyzw vf4,vf4           nop                ;Convert colour to int
sub.w vf5,vf0,vf0            nop                ;Clear w of vertext to zero ( Needed for
drawkick bit )
nop                          waitq              ;Wait for result
mulq.xyz vf5,vf5,Q           nop                ;Final screen XYZ
ftoi4.xyz vf5,vf5            nop                ;Convert vertex to int;
nop                          sq.xyzw vf4,-2(vi1);Store colour over normal;
nop                          sq.xyzw vf5,-1(vi1);Store transformed vertex in place
nop[e]                       nop
nop                          nop
```

V3.0 Performance

- Version 3 draws T-Rex in 27 scanlines
- This is about 37% faster than V2.0
- Performance is now around 4.7 Mvert/s
  - Getting there ;-)

A Tale of Two Dinosaurs - June 1999

Confidential

My 4th renderer

- Microcode is now pipelined with EE
  - While VU0 is processing vertices, EE is building display list

A Tale of Two Dinosaurs - June 1999 Confidential

The main change to the CPU code for this render (compared to version 3) is to use non-interlocking coprocessor reads to transfer results from VU0 while executing a micro subroutine. (I have simplified the postamble for legibility.)

```
asm __volatile__ ( "
la $9,buffer          #GIF buffer
vcallms vu0_first     #Start calculation for first point, no result because of pipeline
or $8,$0,120
vcallms vu0_next      #Calculate 2nd point, no result yet
_calc_next
vcallms vu0_next      #Calculate next point
qmfc2.ni $10,vf8      #Read colour
qmfc2.ni $11,vf9      #Read vector
sq $10,0($9)          #Store previous colour
sq $11,0x10($9)       #Store previous vertex
subu $8,$8,1
add $9,$9,0x20
bnez $8,_calc_next
nop;
vcallms vu0_final     #Complete operation for final point
": : : "$8","$9","$10","$11" );
```

V4.0 Performance

- Version 4 draws T-Rex in 14 scanlines
- Over 2x faster than V3.0
  - Over 5x faster than version 1
- Performance is now 9.75 Mvert/s

A Tale of Two Dinosaurs - June 1999 Confidential

```
vu0_first:
nop                     iaddiu vi1,vi0,0            ;Buffer at 0
nop                     lqi.xyzw vf1,(vi1++)        ;Fetch  normal
nop                     lqi.xyzw vf2,(vi1++)        ;Fetch vertex
itof12.xyzw vf1,vf1     nop                         ;Convert normal to float
itof4.xyzw vf2,vf2      nop                         ;Convert vertex to float
sub.xyzw vf5,vf5,vf5    nop ;Clear w of vertext to zero ( Needed for drawkick bit )
sub.xyzw vf7,vf7,vf7    nop                         ;Clear w of colour to zero
```

In the main loop, an extra stage is added to the pipe, just to copy the results of the second stage to result registers (vf8 and vf9). This allows the CPU to read the results at the start of the routine rather than waiting for the end.

```
vu0_next:
mulax.xyzw ACC,vf24,vf2         nop
madday.xyzw ACC,vf25,vf2        nop
maddaz.xyzw ACC,vf26,vf2        move.xyz vf5,vf3
maddw.xyzw vf3,vf27,vf0         nop
mulax.xyzw ACC,vf16,vf1         nop
madday.xyzw ACC,vf17,vf1        nop
maddz.xyzw vf4,vf18,vf1         nop
mulq.xyz vf5,vf5,Q              lqi.xyzw vf1,(vi1++)
ftoi0.xyz vf7,vf6               lqi.xyzw vf2,(vi1++)
nop                             div Q,vf0w,vf3w    ;Calc 1.0/w for perspective
maxx.xyzw vf4,vf4,vf0           nop
ftoi4.xyz vf5,vf5               nop
itof12.xyzw vf1,vf1             nop
itof4.xyzw vf2,vf2              nop
mulax.xyzw ACC,vf20,vf4         move.xyzw vf8,vf7 ;Copy normal ( 3rd stage of pipe is read of
result )
madday.xyzw ACC,vf21,vf4        move.xyzw vf9,vf5 ;Copy vertex ( ditto )
maddaz[e].xyzw ACC,vf22,vf4     nop
maddw.xyz vf6,vf23,vf0          nop                ;Read final point
```

```
vu0_final:
mulq.xyz vf3,vf3,Q        nop        ;Final perspective point
sub.w vf3,vf0,vf0         nop
ftoi0[e].xyz vf6,vf6      nop        ;Final colour
ftoi4.xyz vf3,vf3         nop
```

My 5th renderer

- Major change!!!
- Renderer code moved to VU1
- Now DMA flows via VIF1 to VU1
- GIF path 1 used
  - no need to build display list on scratchpad

A Tale of Two Dinosaurs - June 1999 Confidential

With this version of the renderer, the CPU is not really involved at all in the transformation, and the main loop just kicks off DMA transforms...

```
// Wait for VU1 idle, ( matrices in registers )
sceDmaSync(dmavif1, 0, 0);
if (swap==0){
tag_vif1[0].p[0] = SCE_VIF0_SET_MSCAL( vu1_transform0,0 );
tag_vif1[0].p[1] = SCE_VIF0_SET_UNPACK(512, (40 *6), V3x16, 0);
// Transfer 40 polyss using  V3-16 format to VIF1_MEM 16->
}else{
tag_vif1[0].p[0] = SCE_VIF0_SET_MSCAL( vu1_transform512,0 );
tag_vif1[0].p[1] = SCE_VIF0_SET_UNPACK(0, (40 *6), V3x16, 0);
// Transfer 40 polyss using  V3-16 format to VIF1_MEM 16->
}
tag_vif1[0].next = &model[poly *((6*3))];
sceDmaSend(dmavif1, spad_vif1);
swap ^= 1;
```

A Tale of Two Dinosaurs - June 1999

Confidential

The vu1 code is split into 3 parts, a preamble, main loop, and postamble. The preamble code handles double buffering. (I have 2 entry points to process verteces in different buffers.)

```
vu1_trans0:
nop                         ibeq vi0,vi0,vu1_trans
nop                         iaddiu vi1,vi0,0      ;Buffer at 0
vu1_trans512:
nop                         iaddiu vi1,vi0,512    ;Start transforming points at quadword 512
vu1_trans:
nop                         iaddiu vi2,vi1,255    ;Build GIF tag at location +256
nop                         iaddiu vi4,vi1,256
nop                         lqi.xyzw vf1,(vi1++);Fetch normal
nop                         lqi.xyzw vf2,(vi1++);Fetch vertex
itof12.xyzw vf1,vf1         nop                   ;Convert normal to float
itof4.xyzw vf2,vf2          nop                   ;Convert vertex to float
sub.xyzw vf5,vf5,vf5        nop                   ;Clear w of vertext to zero ( Needed for
drawkick bit )
sub.xyzw vf6,vf6,vf6        nop                   ;Clear w of colour to zero
nop                         iaddiu vi3,vi0,120    ;40 Triangles x 3 points ( N + Vper vertex )
nop                         nop
```

A Tale of Two Dinosaurs - June 1999

Confidential

The main loop processes the points, using a 2 stage pipeline.

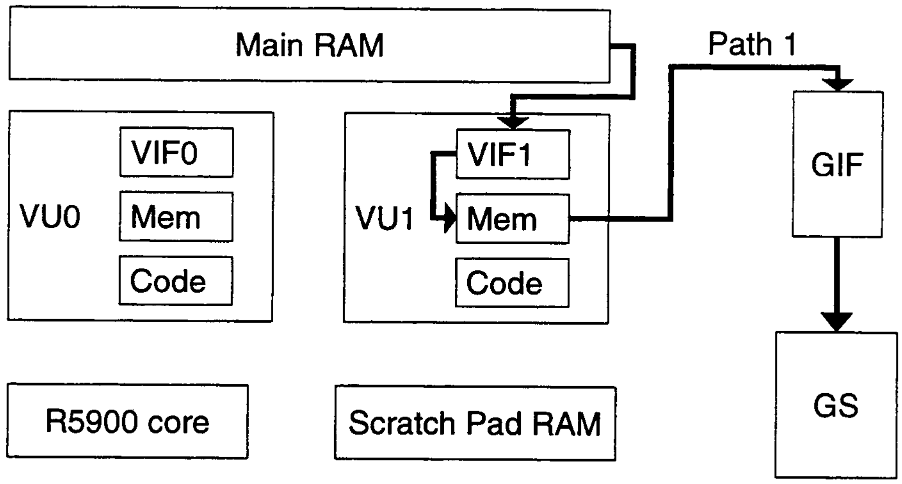
```
vu1_loop:
mulax.xyzw ACC,vf24,vf2         move.xyz vf5,vf3
madday.xyzw ACC,vf25,vf2        nop
maddaz.xyzw ACC,vf26,vf2        nop
maddw.xyzw vf3,vf27,vf0         nop
mulax.xyzw ACC,vf16,vf1         nop
madday.xyzw ACC,vf17,vf1        nop
maddz.xyzw vf4,vf18,vf1         nop
mulq.xyz vf5,vf5,Q              lqi.xyzw vf1,(vi1++)
ftoi0.xyz vf6,vf6               lqi.xyzw vf2,(vi1++)
nop                             div Q,vf0w,vf3w    ;Calc 1.0/w for perspective
maxx.xyzw vf4,vf4,vf0           nop
ftoi4.xyz vf5,vf5               nop
itof12.xyzw vf1,vf1             sq.xyz vf6,0(vi2)
itof4.xyzw vf2,vf2              nop
mulax.xyzw ACC,vf20,vf4         isubiu vi3,vi3,1
madday.xyzw ACC,vf21,vf4        sq.xyz vf5,1(vi2)
maddaz.xyzw ACC,vf22,vf4        ibne vi3,vi0,vu1_loop
maddw.xyz vf6,vf23,vf0          iaddiu vi2,vi2,2      ;Draw transformed polys
```

A Tale of Two Dinosaurs - June 1999 Confidential

The postamble code finishes off the calculations for the final point and then starts the path 1 DMA transfer to the GS (using the xgkick instruction).

```
mulq.xyz vf3,vf3,Q      nop                  ;Final perspective point
sub.w vf3,vf0,vf0       nop
ftoi0.xyz vf6,vf6       nop                  ;Final colour
ftoi4.xyz vf3,vf3       nop
nop                     sq.xyzw vf28,0(vi4);Store GIF tag first
nop                     sq.xyzw vf6,0(vi2)
nop                     sq.xyzw vf3,1(vi2)
nop                     nop
nop                     nop
nop                     nop
nop                     xgkick vi4
nop                     nop
nop                     nop
nop                     nop
nop[e]                  nop
nop                     nop
```

Data path



A Tale of Two Dinosaurs - June 1999

Confidential

V5.0 Performance

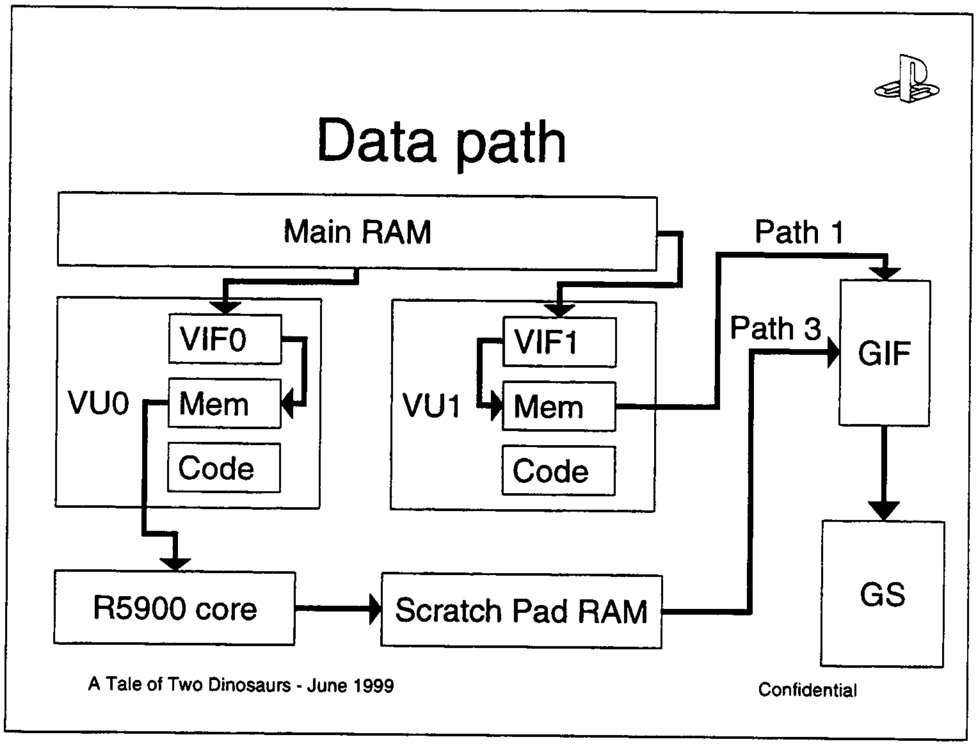
- Version 5 draws T-Rex in 10 scanlines
- About 1.3x faster than V4.0
  - Over 7x faster than version 1
- Performance is now around 12.75 Mvert/s
- Double buffering made the difference

A Tale of Two Dinosaurs - June 1999

Confidential

My 6th renderer

- Run the VU0/EE renderer and VU1 renderer in parallel
- Split the data between VU0/EE and VU1
  - proportional to their rendering speed
  - 43% processed by VU0/EE
  - 57% processed by VU1


Data path
Main RAM
Path 1
VIF0
VU0
Mem
Code
VIF1
VU1
Mem
Code
Path 3
GIF
GS
R5900 core
Scratch Pad RAM
A Tale of Two Dinosaurs - June 1999
Confidential

V6.0 Performance

- Version 6 draws T-Rex in 6 scanlines
  - VU0 performance is 9.75 Mverts/s
  - VU1 performance is 12.75 Mverts/s
  - Total performance therefore 22.5 Mverts/s
  - Polygons for the T-Rex model are being displayed using different data paths

The renderer code for version 4 (VU0/EE) is still not as fast as the VU1 code from version 5, but this can be attributed to the single buffering that is used (for ease of implementation). However both renderers can be operated in parallel.

Because the data paths are different for the two renderers the only bottlenecks are the DMA paths from main memory and the drawing path to the GS. The bandwidth of both is great enough for this not to be a problem.

How fast could it go? (1/3)

- Render operation requires following
  - 5 cycles for vertex transform / perspective
  - 4 cycles for normal rotation / clamping
  - 4 cycles for 3 parallel lights + ambient
  - 4 cycles for int to float & float to int
- Total 17 cycles @ 300Mhz $\Rightarrow$ 17.7 Mvert/s
- VU0+VU1 peak $\therefore$ 35.4 Mvert/s!
  - Remember peak never includes memory costs or DMA operations ;-)

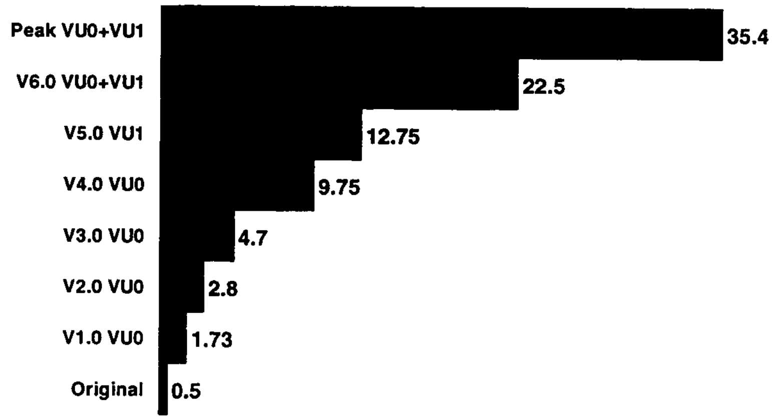
How fast could it go? (2/3)

- GS specifications for drawing are:
  - 37.5 million gourard polys / second (max)
  - 25 million textured lit polys / second
    - 8 x 8 triangles
- In practice, the GS is so overpowered that it will hardly ever be the bottleneck

How fast could it go? (3/3)

- Data format is important for high polys/s
  - int to float conversion takes around 10% of the processing time
  - Formats involving table lookups are not conducive to DMA transfers
  - Meshing the polygons means less vertices to transform, allowing a higher polygon count

Performance comparison



A Tale of Two Dinosaurs - June 1999

Confidential

Conclusions

- VU1 data path to GIF is most efficient
- However performance using EE / VU0 is close when pipelining used
- Parallel execution using both data paths gives best performance with little effort
  - Large models can be split and sent along both data paths to screen

The End

A Tale of Two Dinosaurs - June 1999

Confidential